Behavior Driven Development (BDD) and Continuous Integration & Delivery (CI-CD)

The goal of testing is to increase confidence of stakeholders through evidence

- Dan North (https://dannorth.neta)
Different levels of Testing

• **Unit Tests** focuses on testing a specific unit or component, such as a class method. It confirms that the function is built right.

• **Integration Tests** focuses on testing multiple “Units” together as a group.

• **System Tests** focuses on testing the complete software or application as a whole.

• **Acceptance Tests** are the final level of testing which verifies whether the system’s functionality matches the specification. It confirms whether we have built the right thing.
Overview of BDD

• The Agile methodology implements testing and validation as an ongoing process, via the “as a user” approach. BDD using the same approach, makes it the most suitable practice within Agile.

• BDD focuses on the acceptance criteria from the inception by defining how each feature of the application should behave from the end user’s perspective.

• BDD enables collaboration across all stakeholders: Customer, Dev & QA

• In BDD, the Scenarios are created using DSL (Domain Specific Language) and the Code is written to support the target behavior

• BDD results into faster user acceptance cycles leading to faster deployments
BDD is an abstraction of TDD (Test Driven Development). They both essentially follow the same practices, but, BDD focuses more on the behavior of the application, rather than implementation, using a language where all stakeholders can participate and collaborate.

1. **Describe Features**
2. **Create User Stories for each Feature**
3. **Decompose each User Story into multiple Scenarios**
4. **Create Steps to test each Scenario using DSL (Domain Specific Language)**
   - The tests **Fail** initially
     - *The ones that have never failed have the potential to produce false positives*
   - The code is created/refactored for the test to **Pass**
A typical Agile and BDD based workflow

1. **Customer** provides the requirements
2. **Product Owner** creates the Features which are assigned to a release
3. **Product Owner** and **Scrum Master** create and groom the user stories for the Features
4. The user stories are allocated to the sprint during sprint planning
5. **QA** creates BDD feature files which cover all the scenarios as described in the user story
6. **Developers** create/refactor code for the Feature Files to pass
7. Code and the FFs are committed and tested in local environments (Sanity, Regression, System)
8. The Product Feature is delivered to the Customer’s lab
9. The Feature goes through the User Acceptance Testing (UAT) with the **Customer**
10. The Feature goes live in the Production
## TDD vs BDD

<table>
<thead>
<tr>
<th></th>
<th>TDD (Test Driven Development)</th>
<th>BDD (Behavior Driven Development)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focuses on</td>
<td>developer’s opinion on how</td>
<td>user’s opinion on how they</td>
</tr>
<tr>
<td></td>
<td>functions of the software</td>
<td>want the application to behave</td>
</tr>
<tr>
<td></td>
<td>should work. It is basically</td>
<td>It is basically a customer’s view.</td>
</tr>
<tr>
<td></td>
<td>a programmer’s view.</td>
<td></td>
</tr>
<tr>
<td>A Low level approach</td>
<td></td>
<td>As a user approach</td>
</tr>
<tr>
<td></td>
<td>Verifies whether the</td>
<td>Verifies whether the application</td>
</tr>
<tr>
<td></td>
<td>implementation of the</td>
<td>behaves the way user wants it to</td>
</tr>
<tr>
<td></td>
<td>functionalities are correct</td>
<td>behave</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Benefits of BDD for Program and Release management

- Better collaboration and communication within all stakeholders
- Comprehensible and usable without programming expertise
- The Agile approach has lead to visibly better design and results
- Proven platform for Continuous Integration and Delivery (CI-CD)
The Gherkin language

• The Gherkin language is the Core of any BDD Framework

• It is a Domain Specific, Business Readable language

• Enables all stakeholders to understand the software without a deep technical understanding of the implementation

• In the BDD world, the Test Cases are broken down into the Gherkin Scenarios

• The Scenarios are then implemented using the Gherkin’s **G-W-T** construct

  ✓ **Given** – there is some context
  ✓ **When** – some action occurs
  ✓ **Then** – some result is expected
BDD Framework

Rebaca has extensive experience in the following BDD frameworks.

**Cucumber** (https://cucumber.io/)

- Available in Java, Ruby, JavaScript platforms
- Most popular BDD framework
- Much more vibrant community
- Better reporting support

**Behave** (http://pythonhosted.org/behave/)

- Available in Python platform

**JBehave** (http://jbehave.org/)

- Available in Java platform
### Sample BDD feature file in action

**Feature:** SIP Single Call with RTP  
@ sip-single-call-with-rtp  
**Scenario:** End to end SIP call setup with RTP data transfer

| Given all configured endpoints for SIP are connected successfully |
| When | run SIP INVITE call-flows for SERVER using uas_invite.xml for 1 users configured in user.csv at ABOT |
| When | run SIP INVITE call-flows for CLIENT using uac_invite.xml for 1 users configured in user.csv at ABOT |

**Example SIP DSL grammar**

| When | run the SSH command “cat output file | grep ‘Successful call’ | awk ‘{print $1,$2,$6}’” at ABOT  |
| Then I verify the presence of the following values in the SIP response:  |
| | String | occurrence |
| | Successful call 1 | PRESENT |

**Example SIP and SSH DSL grammar to execute command and validate output**

| When | run the SSH command “cat output file | grep ‘RTP pckts sent’ | awk ‘{print $1,$3,$4,$5}’” at ABOT  |
| Then I verify the presence of the following values in the SSH response:  |
| | String | occurrence |
| | 236 RTP pckts sent | PRESENT |
How BDD helps in Continuous Integration and Delivery

Continuous Integration and Delivery (CI-CD) enables the features to be integrated, tested and deployed into Production on an ongoing basis.

It is possible to do CI-CD without BDD, but with the BDD practices, the benefits are evidently better.

- **BDD ensures that the features are built using the first time right formula**
- **Agile and BDD work together to slice the target features into small shippable user stories and make incremental releases**
- **BDD Provides clear specifications in an executable form**
- **BDD’s “as a user” testing practice yield results with better quality**

**Successful CI-CD**
Continuous Integration (CI) workflow

- The developer creates/refactors the code and unit tests it
- The code is committed into the source control (SVN)
- The build is triggered in the build server (Jenkins) on demand/periodically
- The test beds are upgraded with the new build (CI Server – Jenkins)
- Automated Tests (Sanity, Regression, System) are triggered using a BDD based framework
- Comprehensive drill down reports are published and all stakeholders are notified
Continuous Delivery (CD) workflow

• CI-CD provides **Customer Release Candidate** in an on-going basis

• CI-CD enables instant notification for software, environmental, deployment & operational issues that would have otherwise been identified much late in the lifecycle causing delays

• **BDD + CI-CD enables faster User Acceptance** and **Production release**

**Rebaca** has extensive experience in building a BDD powered framework to implement CI-CD in the software development life cycle providing Automation at each step to create the environment (handle orchestration), deploy the software, perform all configurations, run test suites, generate reports, collect artifacts and notify all stakeholders.